In [4]:
```
# type your code here
#used to perform dataframe related operations
import pandas as pd
#user to perform any mathematical operations
import numpy as np
#visualization
import seaborn as sns
import matplotlib.pyplot as plt
#for scaling
from sklearn.preprocessing import MinMaxScaler,StandardScaler,RobustScaler #use std scaler only when the data is norma
#for transformation
from sklearn.preprocessing import PowerTransformer
#warnings
from warnings import filterwarnings
filterwarnings('ignore')

from sklearn.model_selection import train_test_split
#for performing linear regression
from statsmodels.api import OLS
from sklearn.linear_model import LinearRegression

#for testing performance of model
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_squared_error

#for multicollinearity treatment
from statsmodels.stats.outliers_influence import variance_inflation_factor

9#for testing normality of residuals
from statsmodels.graphics.gofplots import qqplot

from statsmodels.api import add_constant

# import various functions from statsmodel to perform linear regression
import statsmodels
import statsmodels.api as sm
import statsmodels.stats.api as sms
from statsmodels.graphics.gofplots import qqplot
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.tsa.api as smt

# import various functions from scipy
from scipy import stats

# import 'metrics' from sklearn is used for evaluating the model performance
from sklearn.metrics import mean_squared_error

# import StandardScaler for scaling the data
from sklearn.preprocessing import StandardScaler

# functions for forward selection
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.feature_selection import RFE
# functions for linear regression
from sklearn.linear_model import LinearRegression

# functions for  cross validation
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn import preprocessing

from sklearn.linear_model import LinearRegression

# import StandardScaler to perform scaling
from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import MinMaxScaler

# import SGDRegressor from sklearn to perform linear regression with stochastic gradient descent
from sklearn.linear_model import SGDRegressor

# import function for ridge regression
from sklearn.linear_model import Ridge

# import function for lasso regression
from sklearn.linear_model import Lasso

# import function for elastic net regression
from sklearn.linear_model import ElasticNet

# import function to perform GridSearchCV
from sklearn.model_selection import GridSearchCV

from mlxtend.feature_selection import SequentialFeatureSelector as sfs

from sklearn import metrics
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import cohen_kappa_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import tree
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import StackingRegressor
from xgboost import XGBRegressor
```

In [5]:
```python
df = pd.read_csv('Iowa Liquor Sales (Jan 2021-Jan 2022) (1).csv')
```

In [6]:
```python
df
```

Out[6]:

	invoice_and_item_number	date	store_number	store_name	address	city	zip_code	store_location	county_number	co
0	INV-33179700135	04-01-2021	2576	Hy-Vee Wine and Spirits / Storm Lake	1250 N Lake St	Storm Lake	50588.0	POINT (-95.200758 42.65318400000001)	11.0	BUENA
1	INV-33196200106	04-01-2021	2649	Hy-Vee #3 / Dubuque	400 Locust St	Dubuque	52001.0	POINT (-90.666497 42.49721900000001)	31.0	DUBL
2	INV-33184300011	04-01-2021	2539	Hy-Vee Food Store / Iowa Falls	640 S. Oak	Iowa Falls	50126.0	POINT (-93.262364 42.508752)	42.0	HA
3	INV-33184100015	04-01-2021	4024	Wal-Mart 1546 / Iowa Falls	840 S Oak	Iowa Falls	50126.0	POINT (-93.262446 42.503407)	42.0	HA
4	INV-33174200025	04-01-2021	5385	Vine Food & Liquor	2704 Vine St.	West Des Moines	50265.0	POINT (-93.741511 41.580206)	77.0	I
...	...	...	...	...	...	...	...	...	...	
1048570	INV-39816800042	03-09-2021	5868	Brothers Market / Bloomfield	207 E Locust St	Bloomfield	52537.0	POINT (-92.412847 40.752691)	26.0	D
1048571	INV-39790400013	03-09-2021	5145	South Side Food Mart	1101 Army Post Rd	Des Moines	50315.0	POINT (-93.628783 41.526511)	77.0	I
1048572	INV-39806400009	03-09-2021	4585	Casey's General Store #2561 / Farley	306, 1st St SW	Farley	52046.0	POINT (-91.006139 42.439552)	31.0	DUBL
1048573	INV-39782700007	03-09-2021	4795	Walgreens #00359 / Des Moines	2545 E Euclid Ave	Des Moines	50317.0	POINT (-93.568668 41.627702000000006)	77.0	I
1048574	INV-39809100004	03-09-2021	5512	Casey's General Store #3639 / Postville	620 W Tilden St	Postville	52162.0	POINT (-91.580193 43.084432)	3.0	ALLAM

1048575 rows × 24 columns

In [7]:
```python
df.columns
```

Out[7]:
```
Index(['invoice_and_item_number', 'date', 'store_number', 'store_name',
       'address', 'city', 'zip_code', 'store_location', 'county_number',
       'county', 'category', 'category_name', 'vendor_number', 'vendor_name',
       'item_number', 'item_description', 'pack', 'bottle_volume_ml',
       'state_bottle_cost', 'state_bottle_retail', 'bottles_sold',
       'sale_dollars', 'volume_sold_liters', 'volume_sold_gallons'],
      dtype='object')
```

In [8]:
```python
df = df.drop(['invoice_and_item_number', 'date','store_name','address','zip_code', 'store_location', 'county_number',
            'vendor_number','item_number','category','volume_sold_gallons'], axis = 1)
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 13 columns):
 #   Column              Non-Null Count    Dtype  
---  ------              --------------    -----  
 0   store_number        1048575 non-null  int64  
 1   city                1048511 non-null  object 
 2   county              1048511 non-null  object 
 3   category_name       1048575 non-null  object 
 4   vendor_name         1048572 non-null  object 
 5   item_description    1048575 non-null  object 
 6   pack                1048575 non-null  int64  
 7   bottle_volume_ml    1048575 non-null  int64  
 8   state_bottle_cost   1048575 non-null  float64
 9   state_bottle_retail 1048575 non-null  float64
 10  bottles_sold        1048575 non-null  int64  
 11  sale_dollars        1048575 non-null  float64
 12  volume_sold_liters  1048575 non-null  float64
dtypes: float64(4), int64(4), object(5)
memory usage: 104.0+ MB
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
store_number           0
city                  64
county                64
category_name          0
vendor_name            3
item_description       0
pack                   0
bottle_volume_ml       0
state_bottle_cost      0
state_bottle_retail    0
bottles_sold           0
sale_dollars           0
volume_sold_liters     0
dtype: int64
```

In [11]:
```
df1 = df.drop(['store_number','city','county','category_name','vendor_name','item_description'],axis = 1)
```

In [12]:
```
df1.describe()
```

Out[12]:

	pack	bottle_volume_ml	state_bottle_cost	state_bottle_retail	bottles_sold	sale_dollars	volume_sold_liters
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06
mean	1.198848e+01	8.248567e+02	1.126751e+01	1.690189e+01	1.186573e+01	1.610171e+02	9.385574e+00
std	7.881474e+00	5.229357e+02	1.129648e+01	1.694280e+01	3.148000e+01	4.850953e+02	3.787383e+01
min	1.000000e+00	2.000000e+01	8.900000e-01	1.340000e+00	1.000000e+00	1.340000e+00	2.000000e-02
25%	6.000000e+00	3.750000e+02	6.000000e+00	9.000000e+00	3.000000e+00	4.200000e+01	1.500000e+00
50%	1.200000e+01	7.500000e+02	8.980000e+00	1.347000e+01	6.000000e+00	8.952000e+01	4.800000e+00
75%	1.200000e+01	1.000000e+03	1.400000e+01	2.100000e+01	1.200000e+01	1.665000e+02	1.050000e+01
max	6.000000e+01	5.250000e+03	1.949020e+03	2.923530e+03	3.780000e+03	5.643000e+04	6.615000e+03

In [13]:
```
df1.isnull().sum()
```

Out[13]:
```
pack                   0
bottle_volume_ml       0
state_bottle_cost      0
state_bottle_retail    0
bottles_sold           0
sale_dollars           0
volume_sold_liters     0
dtype: int64
```

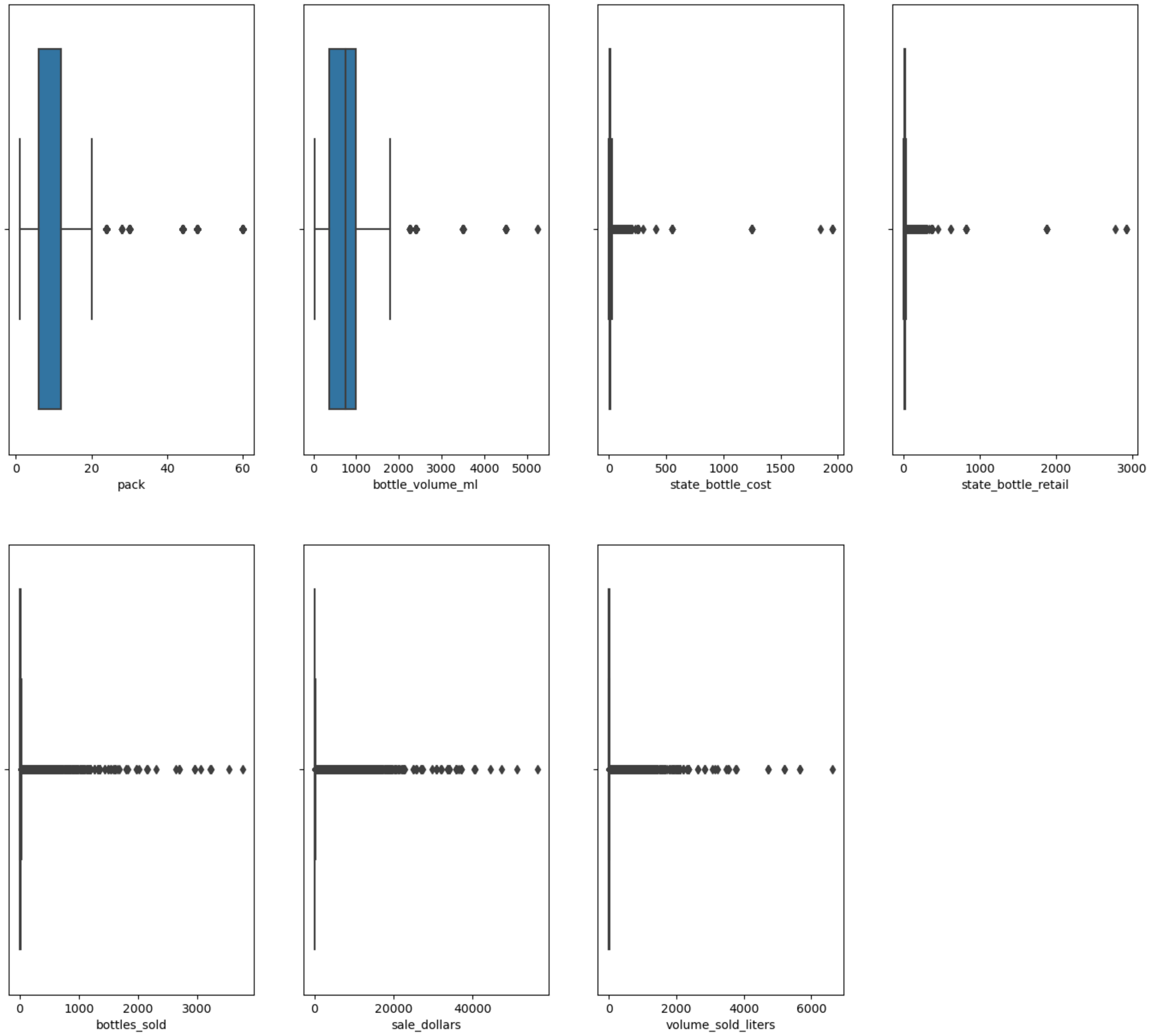
In [14]:
```
df1_clm = df1.columns
```

In [15]:
```
df1_clm
```

Out[15]:
```
Index(['pack', 'bottle_volume_ml', 'state_bottle_cost', 'state_bottle_retail',
       'bottles_sold', 'sale_dollars', 'volume_sold_liters'],
      dtype='object')
```

In [16]:

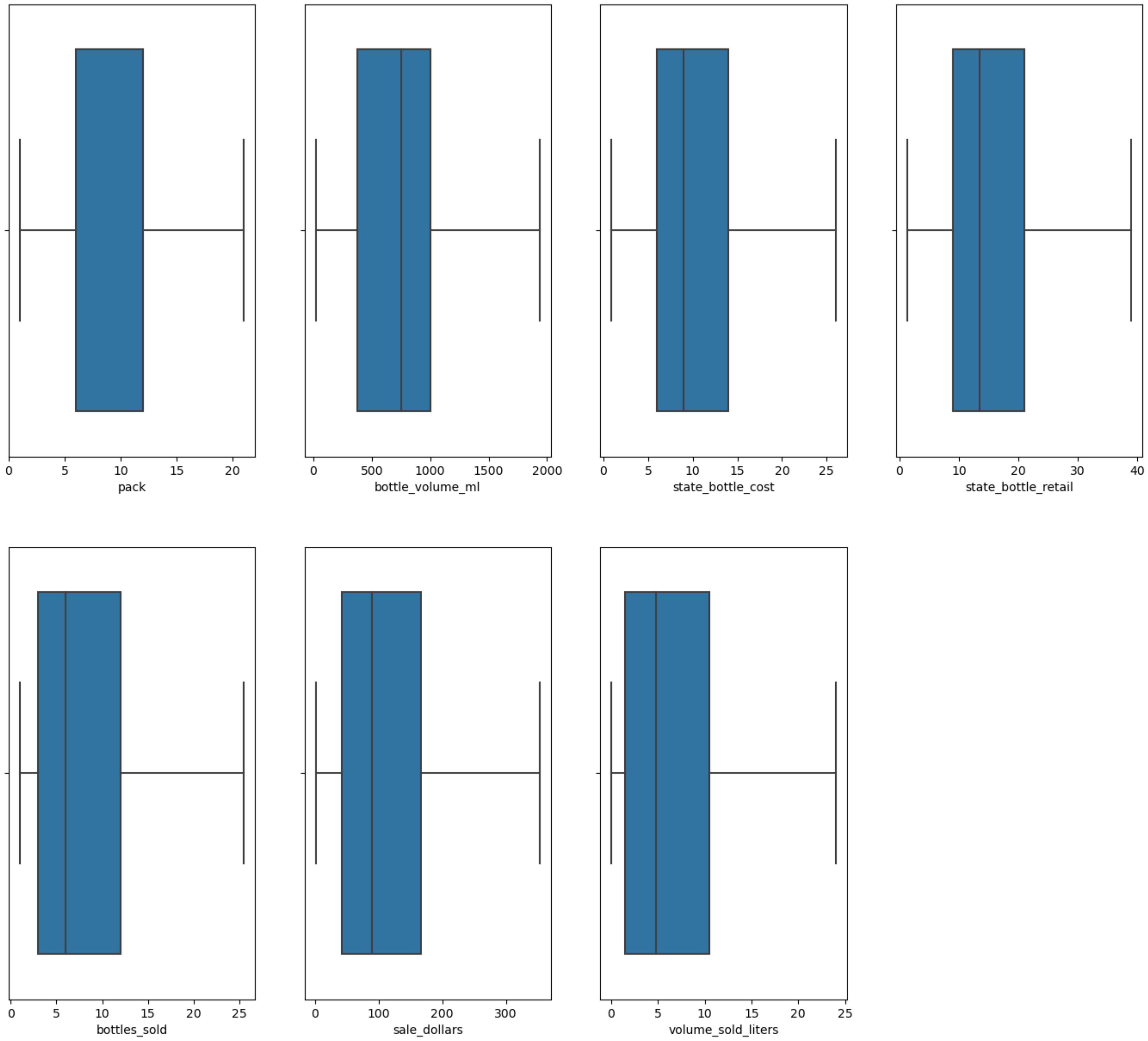
```
t=1
plt.figure(figsize = (17,15))
for i in df1_clm:
    plt.subplot(2,4,t)
    sns.boxplot(df1[i])
    t+=1
plt.show()
```



In [17]:

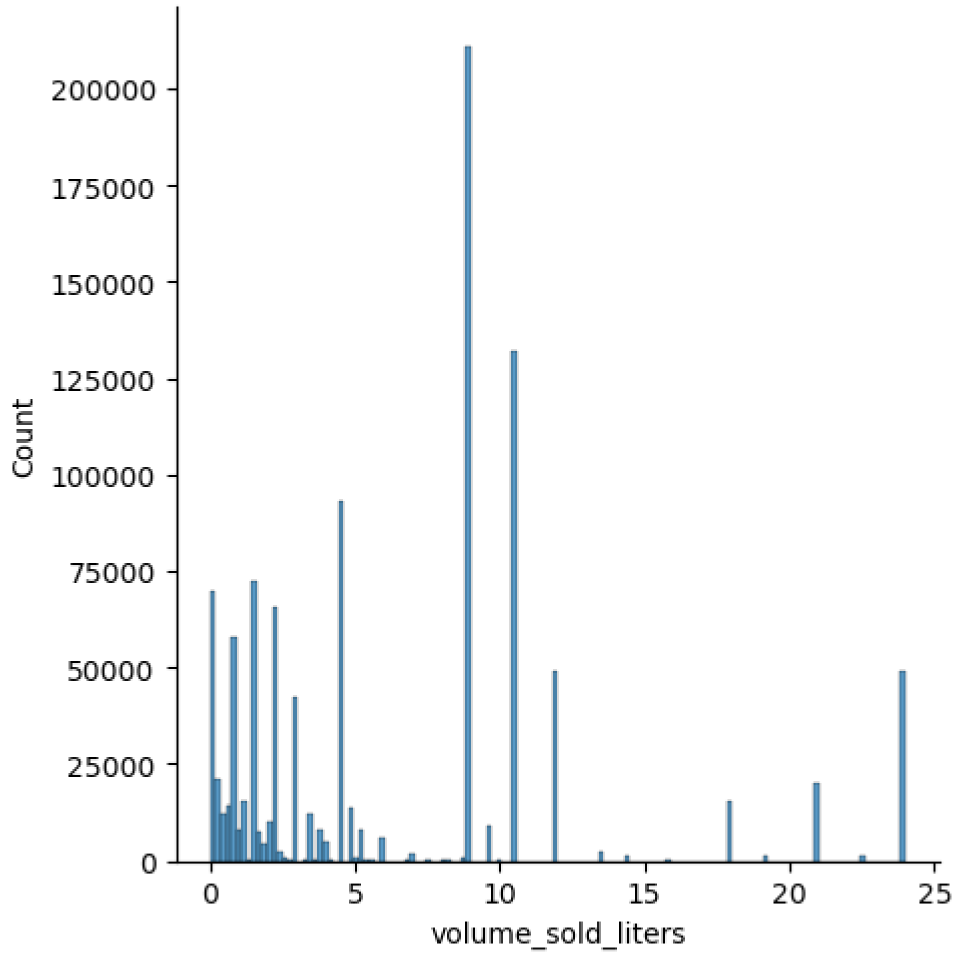
```
for i in df1_clm:
    q1,q3 = np.quantile(df1[i],[0.25,0.75])
    iqr = q3 - q1
    ub = q3 + (1.5 * iqr)
    lb = q1 - (1.5 * iqr)
    df1[i] = np.where(df1[i] > ub, ub, df1[i])
    df1[i] = np.where(df1[i] < lb, lb, df1[i])
```

In [18]:
```
t=1
plt.figure(figsize = (17,15))
for i in df1_clm:
    plt.subplot(2,4,t)
    sns.boxplot(df1[i])
    t+=1
plt.show()
```

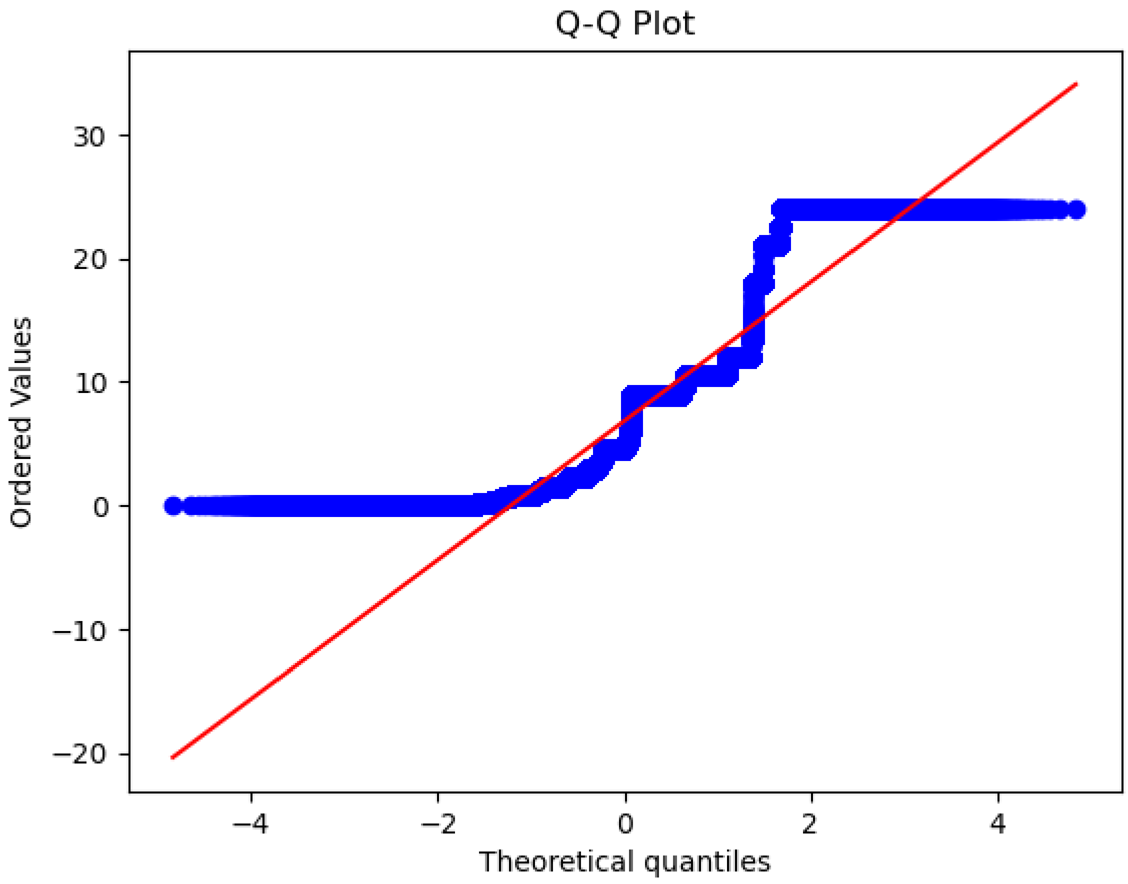


In [19]:
```
sns.displot(df1['volume_sold_liters'], bins = 'auto')
```

Out[19]:
```
<seaborn.axisgrid.FacetGrid at 0x231f79df5b0>
```



In [20]:
```
stats.probplot(df1['volume_sold_liters'], dist="norm", plot=plt)
plt.title('Q-Q Plot')
plt.show()
```



In [21]:
```
stat, p = stats.shapiro(df1['volume_sold_liters'])
alpha = 0.05  # significance level

print("Shapiro-Wilk test statistic:", stat)
print("p-value:", p)

if p > alpha:
    print("The data is normally distributed.")
else:
    print("The data is not normally distributed.")
```

```
Shapiro-Wilk test statistic: 0.8556922674179077
p-value: 0.0
The data is not normally distributed.
```

In [22]:
```
X = df1.drop(['volume_sold_liters'],axis =1)
y= df1.volume_sold_liters
```

In [23]:
```
X = sm.add_constant(X)
```

In [24]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=1,test_size=0.2)
```

In [25]:
```
model_base = sm.OLS(y_train,X_train).fit()
model_base.summary()
```

Out[25]:

OLS Regression Results

Dep. Variable:	volume_sold_liters	**R-squared:**	0.864
Model:	OLS	**Adj. R-squared:**	0.864
Method:	Least Squares	**F-statistic:**	8.853e+05
Date:	Sat, 27 May 2023	**Prob (F-statistic):**	0.00
Time:	00:18:24	**Log-Likelihood:**	-1.8693e+06
No. Observations:	838860	**AIC:**	3.739e+06
Df Residuals:	838853	**BIC:**	3.739e+06
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.3567	0.014	-24.966	0.000	-0.385	-0.329
pack	-0.1457	0.001	-190.315	0.000	-0.147	-0.144
bottle_volume_ml	0.0056	5.74e-06	969.835	0.000	0.006	0.006
state_bottle_cost	-0.0775	0.026	-2.992	0.003	-0.128	-0.027
state_bottle_retail	-0.0824	0.017	-4.767	0.000	-0.116	-0.049
bottles_sold	0.4307	0.001	608.904	0.000	0.429	0.432
sale_dollars	0.0219	5.31e-05	412.285	0.000	0.022	0.022

Omnibus:	84645.774	**Durbin-Watson:**	1.999
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	663140.944
Skew:	-0.135	**Prob(JB):**	0.00
Kurtosis:	7.347	**Cond. No.**	1.25e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.25e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [26]:
```
y_pred = model_base.predict(X_test)
```

In [27]:
```
model_base.rsquared
```

Out[27]: 0.8636209632806905

In [28]:
```
scaler = MinMaxScaler()
mm = scaler.fit_transform(df1)
df_mm = pd.DataFrame(mm, columns= df1.columns)
```

In [29]:
```
X = df_mm.drop(['volume_sold_liters'],axis =1)
y= df_mm.volume_sold_liters
```

In [30]:
```
X = sm.add_constant(X)
```

In [31]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=1,test_size=0.2)
```

In [32]:
```
model_scaled_mm = sm.OLS(y_train,X_train).fit()
model_scaled_mm.summary()
```

Out[32]:

OLS Regression Results

Dep. Variable:	volume_sold_liters	**R-squared:**	0.864
Model:	OLS	**Adj. R-squared:**	0.864
Method:	Least Squares	**F-statistic:**	8.853e+05
Date:	Sat, 27 May 2023	**Prob (F-statistic):**	0.00
Time:	00:18:25	**Log-Likelihood:**	7.9595e+05
No. Observations:	838860	**AIC:**	-1.592e+06
Df Residuals:	838853	**BIC:**	-1.592e+06
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0054	0.001	-9.941	0.000	-0.007	-0.004
pack	-0.1215	0.001	-190.315	0.000	-0.123	-0.120
bottle_volume_ml	0.4455	0.000	969.835	0.000	0.445	0.446
state_bottle_cost	-0.0812	0.027	-2.992	0.003	-0.134	-0.028
state_bottle_retail	-0.1294	0.027	-4.767	0.000	-0.183	-0.076
bottles_sold	0.4400	0.001	608.904	0.000	0.439	0.441
sale_dollars	0.3213	0.001	412.285	0.000	0.320	0.323

Omnibus:	84645.774	**Durbin-Watson:**	1.999
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	663140.944
Skew:	-0.135	**Prob(JB):**	0.00
Kurtosis:	7.347	**Cond. No.**	527.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [33]:
```
scaler = StandardScaler()
ss =  scaler.fit_transform(df1)
df_ss = pd.DataFrame(ss, columns= df1.columns)
```

In [34]:
```
#df_ss
```

In [35]:
```
X = df_ss.drop(['volume_sold_liters'],axis =1)
y= df_ss.volume_sold_liters
```

In [36]:
```
X = sm.add_constant(X)
```

In [37]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=1,test_size=0.2)
```

In [38]:
```
MLR_Score_Card = pd.DataFrame(columns=['Model_Name', 'Alpha (Wherever Required)', 'l1-ratio', 'R-Squared',
                                       'Adj. R-Squared', 'Test_RMSE', 'Test_MAPE'])
def update_MLR_Score_Card(algorithm_name, model, alpha = '-', l1_ratio = '-'):
    global MLR_Score_Card
    MLR_Score_Card = MLR_Score_Card.append({'Model_Name': algorithm_name,
                       'Alpha (Wherever Required)': alpha,
                       'l1-ratio': l1_ratio,
                       'Test_MAPE': get_test_mape(model),
                       'Test_RMSE': get_test_rmse(model),
                       'R-Squared': get_score(model)[0],
                       'Adj. R-Squared': get_score(model)[1]}, ignore_index = True)
```

In [39]:
```
def get_train_rmse(model):
    train_pred = model.predict(X_train)
    mse_train = mean_squared_error(y_train, train_pred)
    rmse_train = round(np.sqrt(mse_train), 4)
    return(rmse_train)
```

In [40]:
```
def get_test_rmse(model):
    test_pred = model.predict(X_test)
    mse_test = mean_squared_error(y_test, test_pred)
    rmse_test = round(np.sqrt(mse_test), 4)
    return(rmse_test)
```

In [41]:
```
def mape(actual, predicted):
    return (np.mean(np.abs((actual - predicted) / actual)) * 100)
def get_test_mape(model):
    test_pred = model.predict(X_test)
    mape_test = mape(y_test, test_pred)
    return(mape_test)
```

In [42]:
```
def get_score(model):
    r_sq = model.score(X_train, y_train)
    n = X_train.shape[0]
    k = X_train.shape[1]
    r_sq_adj = 1 - ((1-r_sq)*(n-1)/(n-k-1))
    return ([r_sq, r_sq_adj])
```

In [43]:
```
model_scaled_ss = sm.OLS(y_train,X_train).fit()
model_scaled_ss.summary()
```

Out[43]:

OLS Regression Results

Dep. Variable:	volume_sold_liters	**R-squared:**	0.864
Model:	OLS	**Adj. R-squared:**	0.864
Method:	Least Squares	**F-statistic:**	8.853e+05
Date:	Sat, 27 May 2023	**Prob (F-statistic):**	0.00
Time:	00:18:27	**Log-Likelihood:**	-3.5445e+05
No. Observations:	838860	**AIC:**	7.089e+05
Df Residuals:	838853	**BIC:**	7.090e+05
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	3.467e-05	0.000	0.086	0.931	-0.001	0.001
pack	-0.1115	0.001	-190.315	0.000	-0.113	-0.110
bottle_volume_ml	0.4776	0.000	969.835	0.000	0.477	0.479
state_bottle_cost	-0.0810	0.027	-2.992	0.003	-0.134	-0.028
state_bottle_retail	-0.1291	0.027	-4.767	0.000	-0.182	-0.076
bottles_sold	0.5046	0.001	608.904	0.000	0.503	0.506
sale_dollars	0.3537	0.001	412.285	0.000	0.352	0.355

Omnibus:	84645.774	**Durbin-Watson:**	1.999
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	663140.944
Skew:	-0.135	**Prob(JB):**	0.00
Kurtosis:	7.347	**Cond. No.**	160.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [44]:
```
lin_reg=LinearRegression()
model_scaled_s=lin_reg.fit(X_train,y_train)
ypred=model_scaled_s.predict(X_test)
rmse=np.sqrt(mean_squared_error(y_test,ypred))
print(rmse)
from sklearn.metrics import r2_score
r2=r2_score(y_test,ypred)
print(r2)
```

```
0.3689946822983233
0.8641077070686953
```

In [45]:
```
update_MLR_Score_Card(algorithm_name = 'Linear Regression (Standard Scaller)', model = model_scaled_s)
MLR_Score_Card
```

Out[45]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.86362	0.369	38.580727

SGD

In [46]:
```
sgd = SGDRegressor(random_state = 10)
linreg_with_SGD = sgd.fit(X_train, y_train)
print('RMSE on train set:', get_train_rmse(linreg_with_SGD))
print('RMSE on test set:', get_test_rmse(linreg_with_SGD))
```

```
RMSE on train set: 0.3696
RMSE on test set: 0.3693
```

In [47]:
```
update_MLR_Score_Card(algorithm_name = 'Linear Regression SGD', model = linreg_with_SGD)
MLR_Score_Card
```

Out[47]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406

Ridge alpha = 1

In [48]:
```
ridge = Ridge(alpha = 1, max_iter = 500)
ridge.fit(X_train, y_train)
print('RMSE on test set:', get_test_rmse(ridge))
```

```
RMSE on test set: 0.369
```

In [49]:
```
update_MLR_Score_Card(algorithm_name ='Ridge Regression (with alpha = 1)', model = ridge, alpha = 1)
MLR_Score_Card
```

Out[49]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702

Ridge Alpha = 2

In [50]:
```
ridge = Ridge(alpha = 2, max_iter = 500)
ridge.fit(X_train, y_train)
print('RMSE on test set:', get_test_rmse(ridge))
```

```
RMSE on test set: 0.369
```

In [51]:
```
update_MLR_Score_Card(algorithm_name ='Ridge Regression (with alpha = 2)', model = ridge, alpha = '2')
MLR_Score_Card
```

Out[51]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702
3	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.3690	38.580676

Lasso

In [52]:
```
lasso = Lasso(alpha = 0.01, max_iter = 500)
lasso.fit(X_train, y_train)
print('RMSE on test set:', get_test_rmse(lasso))
```

```
RMSE on test set: 0.3702
```

In [53]:
```python
update_MLR_Score_Card(algorithm_name ='Lasso Regression', model = lasso, alpha = '0.01')
MLR_Score_Card
```

Out[53]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702
3	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.3690	38.580676
4	Lasso Regression	0.01	-	0.862812	0.862810	0.3702	38.195035

Elasticnet

In [54]:
```python
enet = ElasticNet(alpha = 0.1, l1_ratio = 0.01, max_iter = 500)
enet.fit(X_train, y_train)
print('RMSE on test set:', get_test_rmse(enet))
```

```
RMSE on test set: 0.3754
```

In [55]:
```python
update_MLR_Score_Card(algorithm_name ='Elastic Net Regression', model = enet, alpha = '0.1', l1_ratio = '0.01')
MLR_Score_Card
```

Out[55]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702
3	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.3690	38.580676
4	Lasso Regression	0.01	-	0.862812	0.862810	0.3702	38.195035
5	Elastic Net Regression	0.1	0.01	0.859033	0.859032	0.3754	36.652401

Ridge Hyperparameter Tunning using GridSearchCV

In [56]:
```python
tuned_paramaters = [{'alpha':[1e-15, 1e-10, 1e-8, 1e-4,1e-3, 1e-2, 0.1, 1, 5, 10, 20, 40, 60, 80, 100]}]
ridge = Ridge()
ridge_grid = GridSearchCV(estimator = ridge,
                          param_grid = tuned_paramaters,
                          cv = 10)
ridge_grid.fit(X_train, y_train)
print('Best parameters for Ridge Regression: ', ridge_grid.best_params_, '\n')
print('RMSE on test set:', get_test_rmse(ridge_grid))
```

```
Best parameters for Ridge Regression:  {'alpha': 100}

RMSE on test set: 0.369
```

In [57]:
```python
update_MLR_Score_Card(algorithm_name = 'Ridge Regression (using GridSearchCV)',
                  model = ridge_grid,
                  alpha = ridge_grid.best_params_.get('alpha'))
MLR_Score_Card
```

Out[57]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702
3	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.3690	38.580676
4	Lasso Regression	0.01	-	0.862812	0.862810	0.3702	38.195035
5	Elastic Net Regression	0.1	0.01	0.859033	0.859032	0.3754	36.652401
6	Ridge Regression (using GridSearchCV)	100	-	0.863621	0.863620	0.3690	38.577992

Lasso Hyperparameter Tunning using GridSearchCV

In [58]:
```
# from sklearn.model_selection import RepeatedKFold
# cv1 = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# tuned_paramaters = [{'alpha':[1e-15, 1e-10, 1e-8, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 20]}]
# lasso = Lasso()
# lasso_grid = GridSearchCV(estimator = lasso,
#                           param_grid = tuned_paramaters,
#                           cv = cv1)
# lasso_grid.fit(X_train, y_train)
# print('Best parameters for Lasso Regression: ', lasso_grid.best_params_, '\n')
# print('RMSE on test set:', get_test_rmse(lasso_grid))
```

In [59]:
```
# update_MLR_Score_Card(algorithm_name = 'Lasso Regression (using GridSearchCV)',
#                   model = lasso_grid,
#                   alpha = lasso_grid.best_params_.get('alpha'))
# MLR_Score_Card
```

Elasticnet Hyperparameter Tunning using GridSearchCV

In [60]:
```
# tuned_paramaters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 20, 40, 60],
#                      'l1_ratio':[0.0001, 0.0002, 0.001, 0.01, 0.1, 0.2]}]
# enet = ElasticNet()
# enet_grid = GridSearchCV(estimator = enet,
#                          param_grid = tuned_paramaters,
#                          cv = 10)
# enet_grid.fit(X_train, y_train)
# print('Best parameters for Elastic Net Regression: ', enet_grid.best_params_, '\n')
# print('RMSE on test set:', get_test_rmse(enet_grid))
```

In [61]:
```
# update_MLR_Score_Card(algorithm_name = 'Elastic Net Regression (using GridSearchCV)',
#                   model = enet_grid,
#                   alpha = enet_grid.best_params_.get('alpha'),
#                   l1_ratio = enet_grid.best_params_.get('l1_ratio'))
# MLR_Score_Card
```

Decision Tree

In [62]:
```
decision_tree_Regression = DecisionTreeRegressor(criterion = 'squared_error', random_state = 10)
decision_tree = decision_tree_Regression.fit(X_train, y_train)
```

In [63]:
```
print('RMSE on test set:', get_test_rmse(decision_tree))
```

```
RMSE on test set: 0.0334
```

In [64]:
```
update_MLR_Score_Card(algorithm_name ='decision tree Regression', model = decision_tree)
MLR_Score_Card
```

Out[64]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702
3	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.3690	38.580676
4	Lasso Regression	0.01	-	0.862812	0.862810	0.3702	38.195035
5	Elastic Net Regression	0.1	0.01	0.859033	0.859032	0.3754	36.652401
6	Ridge Regression (using GridSearchCV)	100	-	0.863621	0.863620	0.3690	38.577992
7	decision tree Regression	-	-	0.999179	0.999179	0.0334	0.159738

In [65]:
```
#The 'criterion' parameter of DecisionTreeRegressor must be a str among {'poisson', 'absolute_error', 'friedman_mse',
```

In [66]:
```
# tuned_paramaters = [{'criterion': ['friedman_mse', 'squared_error'],
#                      'max_depth': [None,5,10],
#                      'max_features': ["sqrt", "log2"],
#                      'min_samples_split': [2,5,10],
#                      'min_samples_leaf': [1,2,4],
#                      'max_leaf_nodes': range(1, 10)}]
# decision_tree_Regression = DecisionTreeRegressor(random_state = 10)
# tree_grid = GridSearchCV(estimator = decision_tree_Regression,
#                         param_grid = tuned_paramaters,
#                         cv = 100)
# tree_grid_model = tree_grid.fit(X_train, y_train)
# print('Best parameters for decision tree classifier: ', tree_grid_model.best_params_, '\n')
```

In [67]:
```
# dt_model = DecisionTreeRegressor(criterion = tree_grid_model.best_params_.get('criterion'),
#                                  max_depth = tree_grid_model.best_params_.get('max_depth'),
#                                  max_features = tree_grid_model.best_params_.get('max_features'),
#                                  max_leaf_nodes = tree_grid_model.best_params_.get('max_leaf_nodes'),
#                                  min_samples_leaf = tree_grid_model.best_params_.get('min_samples_leaf'),
#                                  min_samples_split = tree_grid_model.best_params_.get('min_samples_split'),
#                                  random_state = 10)

# # use fit() to fit the model on the train set
# dt_model = dt_model.fit(X_train, y_train)
```

Random Forest

In [68]:
```
rf_regression = RandomForestRegressor(n_estimators = 10, random_state = 10)
rf_model = rf_regression.fit(X_train, y_train)
```

In [69]:
```
print('RMSE on test set:', get_test_rmse(rf_model))
```

```
RMSE on test set: 0.0332
```

In [70]:
```
update_MLR_Score_Card(algorithm_name ='Random Forest Regression', model = rf_model)
MLR_Score_Card
```

Out[70]:

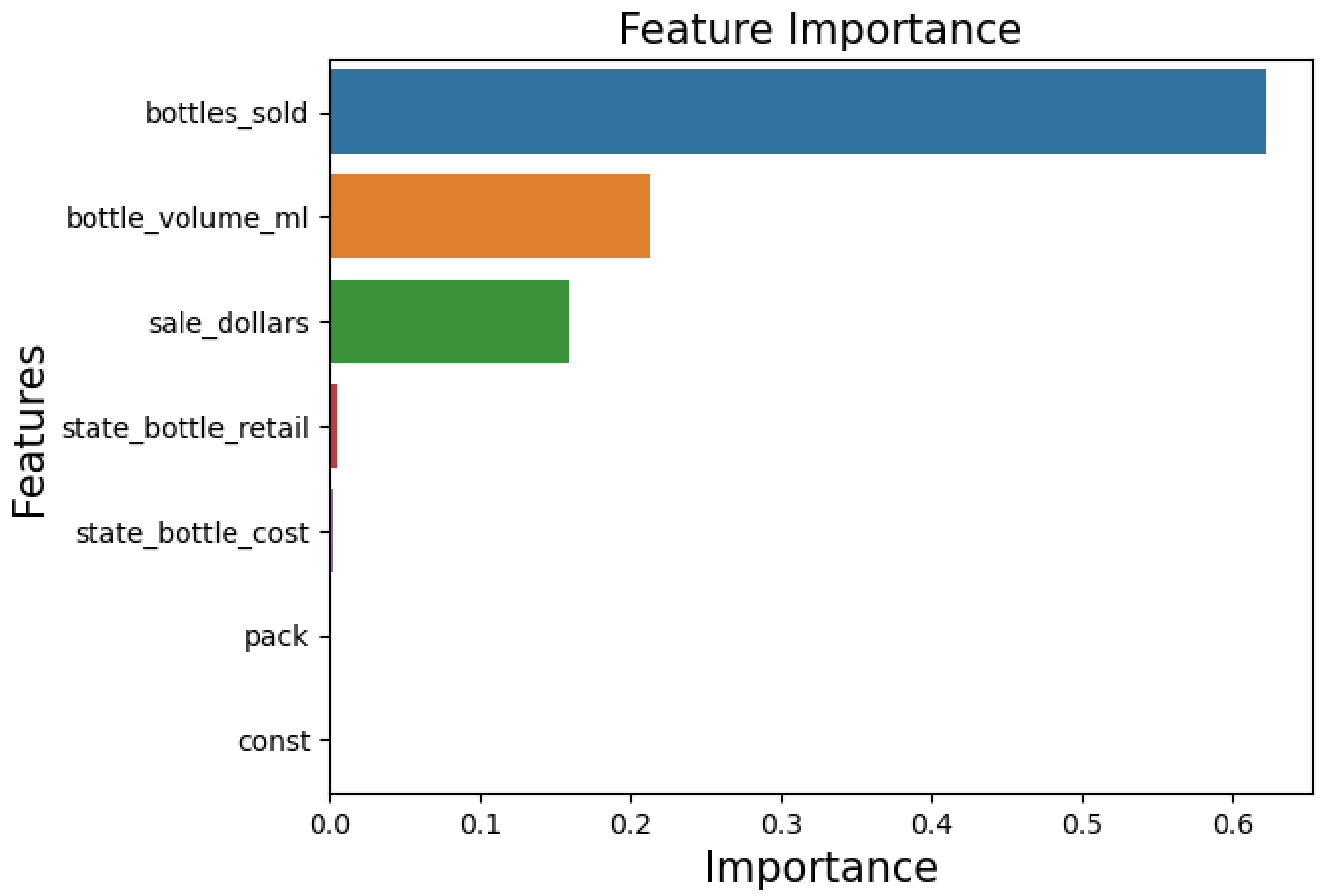
	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702
3	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.3690	38.580676
4	Lasso Regression	0.01	-	0.862812	0.862810	0.3702	38.195035
5	Elastic Net Regression	0.1	0.01	0.859033	0.859032	0.3754	36.652401
6	Ridge Regression (using GridSearchCV)	100	-	0.863621	0.863620	0.3690	38.577992
7	decision tree Regression	-	-	0.999179	0.999179	0.0334	0.159738
8	Random Forest Regression	-	-	0.999167	0.999167	0.0332	0.163628

Random Forest Regression Grid Search CV

In [71]:
```
# tuned_paramaters_rf = [{'criterion': ['friedman_mse', 'squared_error'],
#                      'n_estimators': [10, 30, 50, 70, 90],
#                      'max_depth': [10, 15, 20],
#                      'max_features': ['sqrt', 'log2'],
#                      'min_samples_split': [2, 5, 8, 11],
#                      'min_samples_leaf': [1, 5, 9],
#                      'max_leaf_nodes': [2, 5, 8, 11]}]
# random_forest_Regression = RandomForestRegressor(random_state = 10)
# rf_grid = GridSearchCV(estimator = random_forest_Regression,
#                       param_grid = tuned_paramaters_rf,
#                       cv = 10)
# rf_grid_model = rf_grid.fit(X_train, y_train)
# print('Best parameters for random forest classifier: ', rf_grid_model.best_params_, '\n')
```

In [72]:
```
# rf_model = RandomForestClassifier(criterion = rf_grid_model.best_params_.get('criterion'),
#                                   n_estimators = rf_grid_model.best_params_.get('n_estimators'),
#                                   max_depth = rf_grid_model.best_params_.get('max_depth'),
#                                   max_features = rf_grid_model.best_params_.get('max_features'),
#                                   max_leaf_nodes = rf_grid_model.best_params_.get('max_leaf_nodes'),
#                                   min_samples_leaf = rf_grid_model.best_params_.get('min_samples_leaf'),
#                                   min_samples_split = rf_grid_model.best_params_.get('min_samples_split'),
#                                   random_state = 10)
# rf_model = rf_model.fit(X_train, y_train)
# print('Classification Report for test set:\n', get_test_report(rf_model,test_data = X_test))
```

In [73]:
```
important_features = pd.DataFrame({'Features': X_train.columns,
                                   'Importance': rf_model.feature_importances_})
important_features = important_features.sort_values('Importance', ascending = False)
sns.barplot(x = 'Importance', y = 'Features', data = important_features)
plt.title('Feature Importance', fontsize = 15)
plt.xlabel('Importance', fontsize = 15)
plt.ylabel('Features', fontsize = 15)
plt.show()
```



Ada Boost

In [74]:
```
ada_model = AdaBoostRegressor(n_estimators = 40, random_state = 10)
ada_model1 = ada_model.fit(X_train, y_train)
```

In [75]:
```
print('RMSE on test set:', get_test_rmse(ada_model1))
```

```
RMSE on test set: 0.4107
```

In [76]:
```
update_MLR_Score_Card(algorithm_name ='Ada Boost Regression', model = ada_model1)
MLR_Score_Card
```

Out[76]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702
3	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.3690	38.580676
4	Lasso Regression	0.01	-	0.862812	0.862810	0.3702	38.195035
5	Elastic Net Regression	0.1	0.01	0.859033	0.859032	0.3754	36.652401
6	Ridge Regression (using GridSearchCV)	100	-	0.863621	0.863620	0.3690	38.577992
7	decision tree Regression	-	-	0.999179	0.999179	0.0334	0.159738
8	Random Forest Regression	-	-	0.999167	0.999167	0.0332	0.163628
9	Ada Boost Regression	-	-	0.832122	0.832121	0.4107	57.784979

Gradient Boosting Regression

In [77]:
```
# gboost_model = GradientBoostingRegressor(n_estimators = 150, max_depth = 10, random_state = 10)
# gboost_model.fit(X_train, y_train)
```

XGB Regression

In [78]:
```
xgb_model = XGBRegressor(max_depth = 10, gamma = 1)
xgb_model1 = xgb_model.fit(X_train, y_train)
```

In [79]:
```
print('RMSE on test set:', get_test_rmse(xgb_model1))
```

```
RMSE on test set: 0.034
```

In [80]:
```
update_MLR_Score_Card(algorithm_name ='XGB Regression', model = xgb_model1)
MLR_Score_Card
```

Out[80]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702
3	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.3690	38.580676
4	Lasso Regression	0.01	-	0.862812	0.862810	0.3702	38.195035
5	Elastic Net Regression	0.1	0.01	0.859033	0.859032	0.3754	36.652401
6	Ridge Regression (using GridSearchCV)	100	-	0.863621	0.863620	0.3690	38.577992
7	decision tree Regression	-	-	0.999179	0.999179	0.0334	0.159738
8	Random Forest Regression	-	-	0.999167	0.999167	0.0332	0.163628
9	Ada Boost Regression	-	-	0.832122	0.832121	0.4107	57.784979
10	XGB Regression	-	-	0.999072	0.999072	0.0340	0.679226

Stacking Regression

In [81]:
```
base_learners = [('rf_model', RandomForestRegressor(criterion = 'squared_error', max_depth = 10, max_features = 'sqrt'
                                                     max_leaf_nodes = 8, min_samples_leaf = 5, min_samples_split = 2,
                                                     n_estimators = 50, random_state = 10)),
                 ('Decision_Tree', DecisionTreeRegressor(criterion = 'squared_error', random_state = 10)),
                 ('XGB', XGBRegressor(max_depth = 10, gamma = 1))]

# initialize stacking classifier
# pass the base learners to the parameter, 'estimators'
# pass the Naive Bayes model as the 'final_estimator'/ meta model
stack_model = StackingRegressor(estimators = base_learners, final_estimator = DecisionTreeRegressor(criterion = 'squar

# fit the model on train dataset
stack_model1 = stack_model.fit(X_train, y_train)
```

In [82]:
```
update_MLR_Score_Card(algorithm_name ='Stacking Regression', model = stack_model1)
MLR_Score_Card
```

Out[82]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.3690	38.580727
1	Linear Regression SGD	-	-	0.863352	0.863351	0.3693	38.398406
2	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.3690	38.580702
3	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.3690	38.580676
4	Lasso Regression	0.01	-	0.862812	0.862810	0.3702	38.195035
5	Elastic Net Regression	0.1	0.01	0.859033	0.859032	0.3754	36.652401
6	Ridge Regression (using GridSearchCV)	100	-	0.863621	0.863620	0.3690	38.577992
7	decision tree Regression	-	-	0.999179	0.999179	0.0334	0.159738
8	Random Forest Regression	-	-	0.999167	0.999167	0.0332	0.163628
9	Ada Boost Regression	-	-	0.832122	0.832121	0.4107	57.784979
10	XGB Regression	-	-	0.999072	0.999072	0.0340	0.679226
11	Stacking Regression	-	-	0.998826	0.998826	0.0370	0.169593

In [83]:
```
MLR_Score_Card = MLR_Score_Card.sort_values('Test_RMSE').reset_index(drop = True)
MLR_Score_Card.style.highlight_min(color = 'lightblue', subset = 'Test_RMSE')
```

Out[83]:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Random Forest Regression	-	-	0.999167	0.999167	0.033200	0.163628
1	decision tree Regression	-	-	0.999179	0.999179	0.033400	0.159738
2	XGB Regression	-	-	0.999072	0.999072	0.034000	0.679226
3	Stacking Regression	-	-	0.998826	0.998826	0.037000	0.169593
4	Linear Regression (Standard Scaller)	-	-	0.863621	0.863620	0.369000	38.580727
5	Ridge Regression (with alpha = 1)	1	-	0.863621	0.863620	0.369000	38.580702
6	Ridge Regression (with alpha = 2)	2	-	0.863621	0.863620	0.369000	38.580676
7	Ridge Regression (using GridSearchCV)	100	-	0.863621	0.863620	0.369000	38.577992
8	Linear Regression SGD	-	-	0.863352	0.863351	0.369300	38.398406
9	Lasso Regression	0.01	-	0.862812	0.862810	0.370200	38.195035
10	Elastic Net Regression	0.1	0.01	0.859033	0.859032	0.375400	36.652401
11	Ada Boost Regression	-	-	0.832122	0.832121	0.410700	57.784979

In [ ]: